Methods 79–80 (2015) 3–10

Contents lists available at ScienceDirect

Methods

journal homepage: www.elsevier.com/locate/ymeth

Fast and accurate mapping of Complete Genomics reads

Donghyuk Lee [a,1], Farhad Hormozdiari [b,1], Hongyi Xin [a], Faraz Hach [c], Onur Mutlu [a], Can Alkan [d,*]

[a] Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA
[b] Department of Computer Science, University of California Los Angeles, Los Angeles, CA, USA
[c] School of Computing Science, Simon Fraser University, Burnaby, BC, Canada
[d] Department of Computer Engineering, Bilkent University, Ankara, Turkey

ARTICLE INFO

Article history:
Received 2 May 2014
Accepted 13 October 2014
Available online 22 October 2014

Keywords:
Complete Genomics
Read mapping
Gapped reads
High throughput sequencing

ABSTRACT

Many recent advances in genomics and the expectations of personalized medicine are made possible thanks to power of high throughput sequencing (HTS) in sequencing large collections of human genomes. There are tens of different sequencing technologies currently available, and each HTS platform have different strengths and biases. This diversity both makes it possible to use different technologies to correct for shortcomings; but also requires to develop different algorithms for each platform due to the differences in data types and error models. The first problem to tackle in analyzing HTS data for resequencing applications is the read mapping stage, where many tools have been developed for the most popular HTS methods, but publicly available and open source aligners are still lacking for the Complete Genomics (CG) platform. Unfortunately, Burrows-Wheeler based methods are not practical for CG data due to the gapped nature of the reads generated by this method. Here we provide a sensitive read mapper (sirFAST) for the CG technology based on the seed-and-extend paradigm that can quickly map CG reads to a reference genome. We evaluate the performance and accuracy of sirFAST using both simulated and publicly available real data sets, showing high precision and recall rates.

1. Introduction

High throughput sequencing (HTS) technologies continue to evolve in a fascinating speed since their first use for paired-end sequencing in [1], and whole genome shotgun sequencing (WGS) [2]. Many sequencing technologies were developed, including the Roche/454 [3], Illumina [4], AB SOLiD [5], and Pacific Biosciences [6], among others [7]. Many tools for read mapping [8], variant calling [9, 10], and genome assembly [11] have been developed for most of these platforms.

The availability of many different algorithms and their open source implementations helps researchers take more advantage of the data generated by the HTS technologies. However, there are no publicly available algorithms devised to map and analyze data generated by the Complete Genomics [12] (CG) platform. This is mainly due to the fact that the Complete Genomics company offers only sequencing service, and provides its own analysis results to their customers through the use of proprietary software. The only set of algorithms devised to analyze CG data are described by a group of scientists from the company [13], but the implementations of these algorithms remain proprietary and unreleased. Without doubt, the company's own analysis pipeline is fine tuned for data with different characteristics. Still, research projects that incorporate new algorithm development with the analysis of many genomes, such as the 1000 Genomes Project [14], may benefit from the availability of open-source tools and algorithms to further improve the analysis results. Having an open source tool to analyze CG data can enable other researchers to develop new findings that are otherwise not possible to discover.

In this paper, we present a new read mapper, sirFAST, designed for the data generated by the CG platform. Complete Genomics reads present characteristics different from all other HTS technologies, where a read is composed of several subreads (read sections) that may either overlap, or have gaps between each other. We designed sirFAST to efficiently align such reads to the reference assembly, at the presence of the flexible "expected" gaps/overlaps within each read. Due to the fact that the Burrows-Wheeler transform [15] based methods do not scale well with indels, we implemented sirFAST as a hash based seed-and-extend algorithm. Similar to its Illumina and SOLiD based counterparts [16, 17, 18, 19], sirFAST supports both SAM [20] and DIVET [21] file formats. The implementation of sirFAST is publicly available at http://sirfast.sf.net

* Corresponding author.
E-mail addresses: onur@cmu.edu (O. Mutlu), calkan@cs.bilkent.edu.tr (C. Alkan).
[1] Joint First Authors.

http://dx.doi.org/10.1016/j.ymeth.2014.10.012
1046-2023/

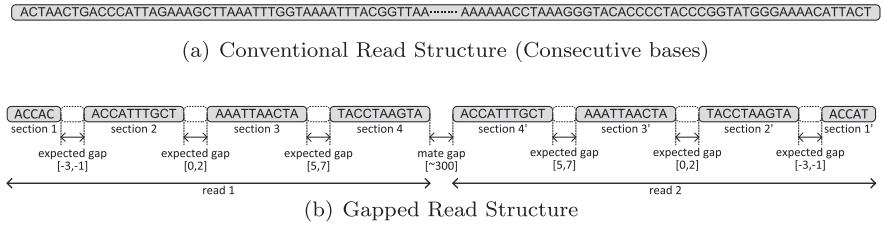


Fig. 1. Conventional read structure vs. Gapped read structure. Note that the read section structures of paired-end reads generated by the Complete Genomics platform are symmetric to each other. Read structure of more recent CG data is changed slightly, however, it is straightforward to provide support for the newer data type using the same algorithms presented in this manuscript.

2. Observation and problem

2.1. Unique characteristics of Complete Genomics reads

The Complete Genomics (CG) platform is based on DNA nanoball technology. Similar to the reads generated by other DNA sequencing technologies, CG provides paired-end reads where each read is 29–35 base-pairs (*bps*), and the fragment size (distance between the two ends) can be between 400 to 1500 bps.

In contrast with the other DNA sequencing technologies that generate reads with consecutive bases (typically 100–8000 bps depending on the platform, as shown Fig. 1(a)), a CG read shows a *gapped read structure* that consists of groups of consecutive bases denoted as *read sections*. There may be either gaps or overlaps between each read sections, denoted as *expected gaps*. Overlaps between read sections are represented as negative gaps. As shown in Fig. 1(b), each read includes four read sections of lengths 5, 10, 10 and 10 bases (35 bps length in total) respectively. Between these read sections, there are expected gaps with sizes that follow a normal distribution with specific mean values (-2, 1 and 6 respectively). As a result, the typical composition of a CG read consists of the following: (i) **first read Section (5 bps)**, (ii) expected gap ($[-3, -1]$ bps), (iii) **second Section (10 bps)**, (iv) expected gap ($[0, 2]$ bps), (v) **third Section (10 bps)**, (vi) expected gap ([5,7] bps), and (vii) **fourth Section (10 bps)**. More recent CG data sets now support slightly different read structure, where 29 bp reads are divided into three read sections of size 10 bp, 9 bp, and 10 bp, respectively[2]. Note that, this difference in the read structure requires only small changes in the implementation of our algorithm, and support for such reads are implemented through a parameter in sirFAST. In the remainder of the paper, we explain our mechanisms based on the assumption that the reads are in the former structure. This is mainly due to the lack of availability of publicly released data generated in this structure. However, we provide simulation-based test results in Table 2.

2.2. The challenge of mapping CG reads

Two paradigms for mapping reads to the reference genome are (1) seed-and-extend, and (2) Burrows-Wheeler Transformation with FM-indexing (BWT-FM) methods [8]. Unfortunately, all existing mappers are designed for mapping consecutive reads to the reference genome, thereby facing challenges or requiring much longer time to map the reads in a gapped read structure (as in CG). Specifically, mapping CG reads with BWT-FM is a harder problem since the expected gaps in CG reads show themselves as indels in an alignment, which increases the run time of BWT-FM search exponentially. Most of the currently available BWT-FM based aligners perform gapped alignment *per read*; instead, they anchor one end with a gap-free alignment, and apply gapped alignment for its mate in regions of close proximity. It is therefore impractical to use BWT-FM based approach for CG reads as both ends contain expected gaps. Although BWT-FM methods outperform seed-and-extend methods in aligning consecutive basepair reads with very low sequence errors, seed-and-extend techniques scale better with high error rate, especially when the errors are indels. We therefore designed our mapper to use the seed-and-extend technique similar to several others available in the literature [8].

Fig. 2 shows the typical operation of a seed-and-extend mapper. The mapper first selects a set of short subsequences of a read as seeds, and quickly queries their locations in the genome using a hash table or a similar data structure. The mapper then tries to extend these initial seed locations by calculating the alignment of the full read against pieces of reference genome at these seed locations. This extension step is commonly called *verification* where the *similarity score* is calculated between the read and the reference. Each insertion, deletion and substitution has a positive score while each matching base has a zero score. A smaller score denotes higher similarity between the read and the reference. The alignment algorithms used in this step are dynamic programming algorithms which tackles insertions and deletions. The complexity of the verification step per seed location is quadratic, which can be reduced to $O(kn)$ if k is an upper bound for allowed indels in the alignment. In the remainder of the paper, we refer to the alignment algorithms as *bp-granularity mapping algorithms*.

3. Methods

We now explain how we map the reads generated by the CG platform to the reference genome using the seed-and-extend methodology (Fig. 3).

- **Selecting Seeds.** The performance and accuracy of seed-and-extend mappers depend on how the seeds are selected in the first stage. The seeds should be selected in a way that guarantees: (1) very high sensitivity to make sure real mapping locations are captured, (2) high specificity to avoid unnecessary alignments, (3) fast seed placement, preferably in $O(1)$ time, and (4) low memory usage. The two main types of seeds are (1) *consecutive seeds* as used in BLAST [22], and (2) *spaced seeds*, first defined in PatternHunter [23]. Spaced seeds carefully select bases from a larger consecutive region of the read to form seeds with gaps in them in order to reduce seed locations. Spaced seeds are considered to be more powerful in terms of higher sensitivity and specificity than consecutive seeds as they provide fewer locations to verify. However, spaced seeds are not suitable for CG reads due to the lack of large consecutive regions in the read (the lengths of expected gaps are flexible and the read sections are only 10-base-long). We, therefore, use *consecutive seeds* of length 10, and implement our seed index using a

[2] http://media.completegenomics.com/documents/DataFileFormats_Standard_Pipeline_2.5.pdf

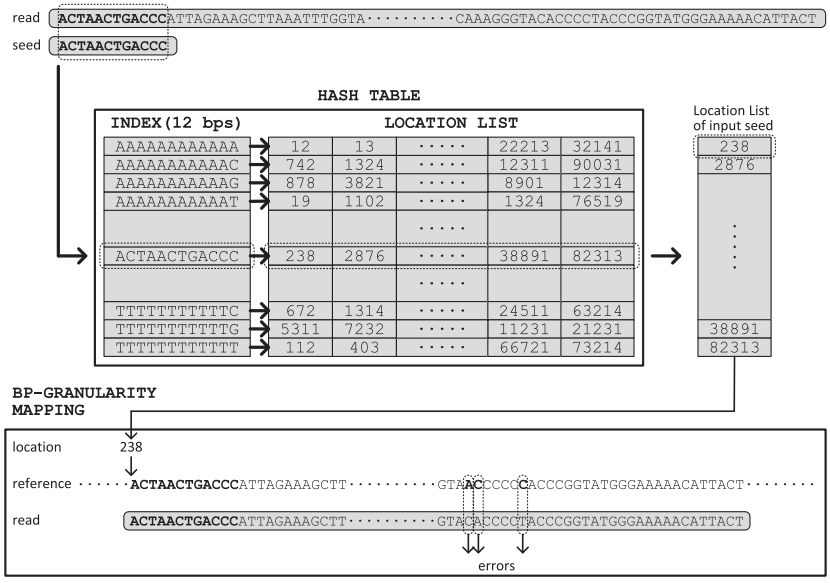


Fig. 2. Mapping consecutive reads using conventional hash-table based mapper.

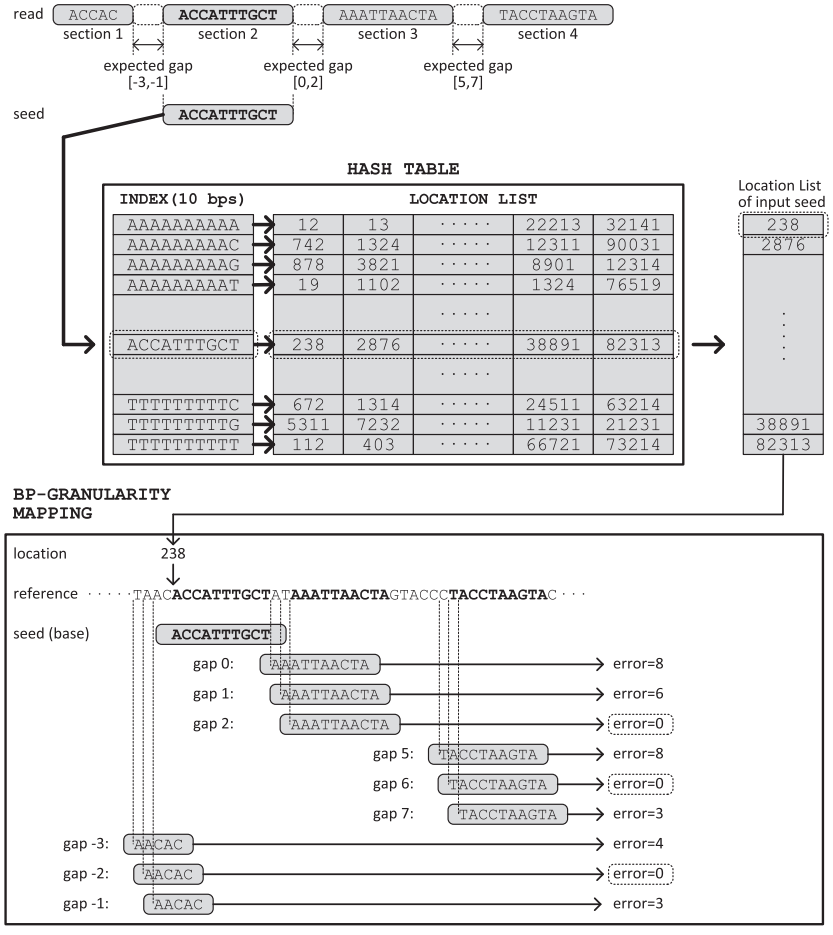


Fig. 3. Mapping gapped read structure using conventional hash-table based mapper.

traditional hash table. We also develop the concept of "combined seeds" to effectively reduce seed locations, since the seed length of 10 bp will yield too many initial locations due to the repetitive nature of the human genomes.

- **Verification.** The seeds serve as a first-step quick filter to find potential locations for the reads, which then have to be verified with an alignment algorithm for the full-length read. This can be done using the Levenshtein [24] or Smith-Waterman [25]

algorithms or their variants. Although these originally $O(n^2)$ algorithms can be accelerated to $O(kn)$ using Ukkonen's method [26], the high number of expected gaps (up to 12 bp) limit the run time improvements. We, therefore take a slightly different approach, and use the Hamming distance in our verification while keeping track of the expected gaps in the alignment.

3.1. Problem

Verifying CG reads is more computationally expensive than verifying conventional reads of consecutive bases. This is because CG reads have gaps of flexible lengths between read sections. Given a *seed location* of one of the read sections, we cannot deduce the exact locations of other read sections from the same read. Rather, we know that they belong to a few *short ranges*. For each read section other than the seed, there exist multiple possible mappings of it within a short range. Therefore, for each *seed location*, we need to verify potential mappings of read sections under all combinations of expected gaps in order to guarantee full sensitivity. The similarity score of a CG read to a reference, is calculated as the sum of the minimum similarity scores of all read sections. This is computationally expensive because for each seed location it requires multiple verifications of many potential mappings of all read sections.

For example, in Fig. 3, if we use the second read section (first 10-bp read section) as the seed, then there are three possible mappings respectively for the first and the third read section with different expected gaps between the second read section. For each possible mapping of the third read section, there exist three more possible mappings of the forth read section with different expected gaps from the third one. As a result, there exists $3 \times 3 \times 3 = 27$ possible combinations of expected gaps between all four read sections. In this example, the minimum similarity score of the read is obtained with a gap of -2 between the first and the second read sections, a gap of 2 between the second and the third read sections and a gap of 6 between the third and the forth read sections (circled in boxes with "error = 0").

Our goal is to build a mechanism that reduces this large search space of possible combinations of expected gaps in the CG read to improve mapping performance while maintaining the mapping sensitivity.

3.2. Combined seeds

As shown in the previous section, CG reads create a large search space of possible combinations of expected gaps between read sections. This search space grows exponentially with the number of expected gaps in the read. To help eliminate this burden, we propose the idea of *combined seeds*, which drastically reduces the search space of CG reads. The key idea is to *merge any two of the 10 bp read sections together as a single combined seed including the gap between them*. With this mechanism, the number of expected gaps in a gapped CG read is reduced from three to two (For example, if we examine Fig. 4, we see that there are originally three expected gaps in the read. By combining Section 2 and 3, the gap between these two seeds would be eliminated). This reduces the search space of the CG read significantly. This combined seed mechanism is based on two key observations: (1) *the distance of mapping locations of two read sections in the reference genome should be within the size of the expected gap* and (2) *the hash table contains information about the relative distances of any two seeds*. Therefore, by obtaining the locations of two read sections from the hash table and comparing them to each other, we can easily determine whether the distance between two read sections are within the range of the expected gap. (We will show an example of this below.).

Although we can combine all three read sections of a CG read together, we choose only two of them at a time in order to maintain high sensitivity (i.e., if three sections are combined, we cannot find all the errors). We require at least two 10-bp read sections matches that follow the aforementioned rule to invoke the verification step. Note that, for a 35 bp read, allowing up to 5% substitutions (a typically used cutoff) will mean a maximum Hamming distance of 2. Hence, by the pigeonhole principle, unless the two allowed substitutions are in two different 10 bp read sections, this seeding step will still provide 100% sensitivity.

We now explain the detailed operation of our *combined seed* technique. The location list of the combined seed is the subset of the location lists of two seeds that can be combined. Fig. 4 shows the mechanism to combine seeds. First, we determine the number of seeds within a read that can be potentially merged into a combined seed (two seeds in Fig. 4). Next, we select two 10 bp read sections as seeds (Section 2 and 3 in Fig. 4). We then query the hash table to load the location lists of the two seeds, and calculate the location list of the combined seed. In this merging step, we use the information in the CG protocol [13] regarding values of the expected gap sizes between read sections. In Fig. 4, we show how to merge read Sections 2 and 3, where the expected gap size is in the range [0,2], and the seed length is 10 bp. Therefore, the map locations of these two seeds in the reference genome should be separated by 10 to 12 basepairs (10 + [0,2]: length of seed + expected gap). Finally, we calculate the effective length of the combined seed as the summation of the lengths of two seeds and the size of the gap in between. Note that a 2-tuple combined seed where the individual seed length is 10 bp, and expected gap size is in the range [0,2] is similar to creating three spaced seeds [23] with weight 20 and lengths 20, 21, and 22. Such spaced seeds are in the form $1^{10}1^{10}$, $1^{10}01^{10}$, and $1^{10}001^{10}$ ($1^{|seed|}$ for seeds, $0^{|gap|}$ for gaps between seeds). Similarly, combined seeds of Sections 3 and 4 can be denoted as a combination of $1^{10}0^{5}1^{10}$, $1^{10}0^{6}1^{10}$, and $1^{10}0^{7}1^{10}$. Finally, we would need to create 9 different forms of spaced seeds to represent the combined seeds of Sections 2 and 4.[3] Due to these issues with spaced seeds, we instead develop and use the combined seeds approach.

There are two main advantages that combined seeds offer in mapping CG reads. They help reduce the potential locations, and also reduce the overhead introduced by the flexible-sized expected gaps. The number of potential mapping locations of the combined seed is drastically reduced when compared to a single seed, because the location list of a combined seed is the intersection of the potential mapping locations of two different seeds. Therefore, mapping using the combined seed decreases the mapping time. Additionally, the expected gaps between the combined sections are automatically detected and removed from consideration in the verification step.

3.3. Multiple combinations of merging seeds

Given a seed length of 10 bp, we can potentially use three seeds in a CG read that are separated by expected gaps. Therefore, to satisfy the high sensitivity constraint, we can merge three different pairs of read sections to generate combined seeds.

3.4. Mapping CG reads using combined seeds and Hamming distance calculation

The entire mapping procedure is shown in Fig. 3, with the exception of using the combined seeds. To describe this briefly,

[3] Note that the negative gaps (overlaps) can be represented in a similar fashion with shorter seeds. For example if the expected gap distribution is within [-2,0] between two 10 bp read sections, then corresponding spaced seeds would be 1^{18}, 1^{19}, and 1^{20}.

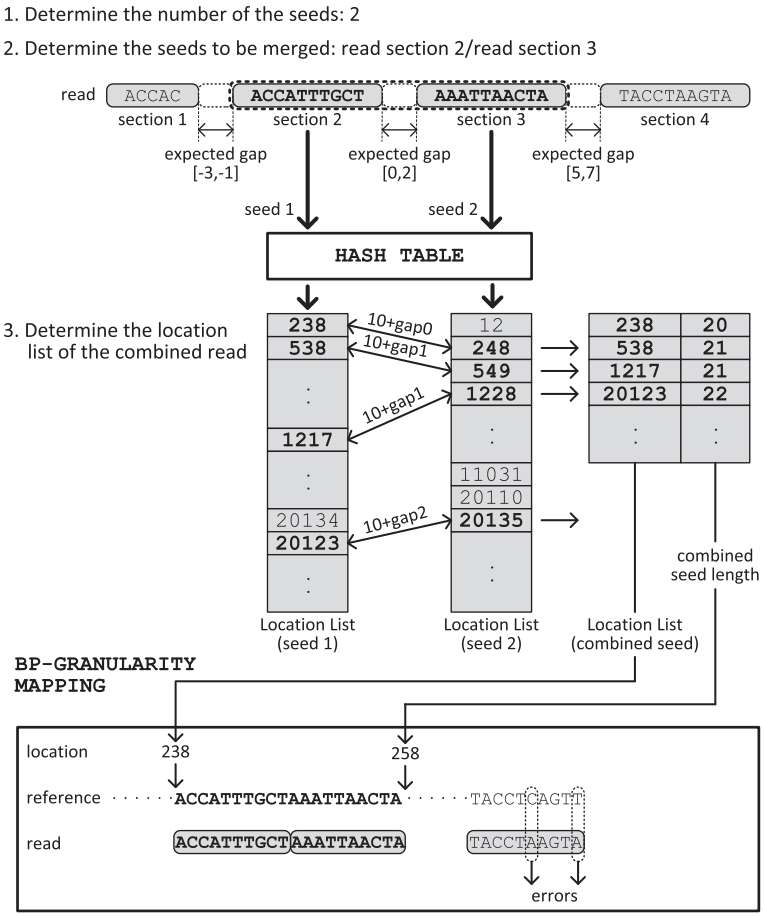


Fig. 4. Mechanism to merge the seeds for generating the combined seed.

we map a CG read by doing the following steps: (1) finding and merging seed locations to generate combined seeds and list of their locations, (2) verifying the map location of the full-length read through Hamming distance calculation [27]. In the verification step, we apply the Hamming distance algorithm to different read sections using a sliding window technique to ensure that we cover all possible combinations of possible expected gaps. The Hamming distance test is not performed on the read sections that are merged into a combined seed for that location, since the hash table and the combined seed procedure guarantees perfect matches for these sections.

4. Results

We used both simulated and real data sets to assess the performance of our methods. We first generated three simulated sets of reads using the human reference genome assembly GRCh37 (hg19), by selecting one subsequence of length 5 bp, and three with 10 bp, in order, and separated with the expected gaps as previously described [12, 13] (Fig. 1(b)). We also repeated this procedure with three 10 bp subsequences followed with a 5 bp subsequence to construct the second end of matepairs. The first set of reads we generated assumed no sequencing errors compared to the reference genome, and the second and third sets included randomly inserted single base mutations at 1% and 2% probability, respectively. For the real data set experiment, we selected one entire slide of reads from the genome of NA12878 that was sequenced by the Complete Genomics corporation, and released publicly [4]. Note that, since there are no other available tools to map CG data, and the official CG mapper is unreleased, we cannot provide performance comparisons against other read mappers. However, in the case of real data set we compared sirFAST with the available mapping result which is released on the CG website.

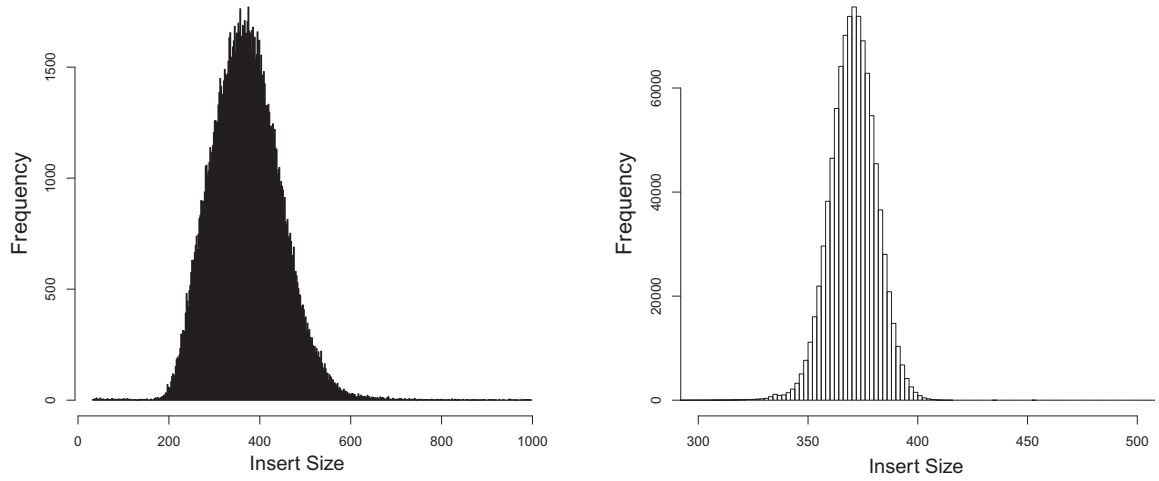
4.1. Mapping reads independently

We first tested sirFAST by mapping all reads independently, without using the matepair information (Table 1) with three different Hamming distance cut offs of 0 to 2 bp. Our aim here was to

Table 1
The performance of sirFAST with three simulated data sets. Each data set included 1 million reads. Note that 1% of the simulated reads in set 1 have one error respect to the reference, thus the best performance is to map 99% of the reads when the error tolerance of sirFAST is set to zero. In the second set, 2% of the reads have one error, therefore the best possible mappability ratio is 98% when the error tolerance is set to zero. Bold numbers show perfect sensitivity (i.e. 100% reads mapped).

Error	Data set	Time (min.)	Map locations	Reads mapped (%)
$e = 0$	Simulated set 1	185	169,840,573	**100**
	Simulated set 2	187	166,584,186	99
	Simulated set 3	189	164,985,213	98
$e = 1$	Simulated set 1	220	870,605,211	**100**
	Simulated set 2	220	860,656,526	**100**
	Simulated set 3	222	852,321,616	99.4
$e = 2$	Simulated set 1	265	1,816,245,093	**100**
	Simulated set 2	265	1,802,365,053	**100**
	Simulated set 3	262	1,783,516,970	**100**

[4] http://www.completegenomics.com/public-data/69-Genomes/



(a) Insert size of real data set (b) Insert size of simulated data set

Fig. 5. The histogram of insert size obtained from mapping the reads in paired-end and single-end mode. The *x*-axis is the insert size and the y-axis is the number of paired-end mapping (frequency) having that insert size.

demonstrate the mapping speed and comprehensiveness. sirFAST was able to map *all* simulated reads in both first and second sets when allowing same number of errors to the generated errors ($e = 0$, $e = 1$, $e = 2$ for set 1, set 2, set 3, respectively), showing full mappability.

In Table 1, we also report the total number of mapped locations to demonstrate the comprehensiveness and potential use of sirFAST for duplication detection [16]. Note that since we treated each read independently in this test, we did not calculate mapping accuracy, which is reported in the next section.

4.2. Paired-end read mapping and precision/recall tests

As a second test to demonstrate the high precision[5] and recall rates[6] of sirFAST mapping, we mapped our simulated and real data sets in paired-end mode. The size distribution of the inferred fragment lengths followed a Gaussian-like distribution as expected in both real (Fig. 5(a)) and simulated (Fig. 5(b)) data sets.

We used the simulated read set mapping result also to estimate the precision and recall rates of sirFAST. Since sirFAST can track all possible mapping locations for each read, and all pairwise combinations of paired-end reads, *theoretically*, the precision and recall rates can be 100%. In fact, our single-end mapping test as explained in Section 4.1 demonstrates the perfect recall. However, although it is powerful to keep track all possible locations for structural variation discovery (SV) [16, 21], it is not practical to do so for other purposes, such as SV discovery. We, therefore, implemented a strategy to select a single "best" mapping, and only report the multiple map locations for read pairs that are candidates for structural variation. We consider the paired-end read mapping location as "best" if:

1. the orientation of the end reads are as expected by the library specifications (i.e., no inversion or tandem duplication candidates).
2. the inferred fragment size is closest to the library mean compared to other mapping alternatives of the same read pair.
3. the total number of mismatches in the mappings of both end reads is minimal among all mapping alternatives of the same read pair.

We consider a read pair to be *concordant* (w.r.t. library specifications) if its orientation is as expected and the inferred fragment size is within the user specified minimum and maximum cutoff values (typically $\mu \pm 4 \times \sigma$). All other read pairs are denoted as *discordant*. We also select a "best" mapping location for discordant pairs, but for such pairs we also allow them to be in inverted orientation. In addition, all possible map locations for the discordant pairs are reported in a separate file.

Using the best mapping locations as assigned by sirFAST, we calculated the precision and recall rates (Table 2). We mapped 500,000 simulated pairs of reads to the human reference genome, and compared the sirFAST-assigned best locations with the actual locations where the reads were selected from. We repeated the experiment on three simulated data sets, demonstrating high precision (>96%) and recall (>99%). The very low false negative rate (<4%) can be explained by the read pairs that, incorrectly, were mapped to transchromosomal locations, that are not reported in the output file in the current version of sirFAST.

4.3. Mapping a real dataset and comparison against the CG mapper

We show comparison between the mapping results generated by sirFAST and the CG mapper using a real data set is in Table 3. We first downloaded 5,360,139,478 reads and their mapping output provided by the company from the Complete Genomics public data archive (URL provided above). We remapped the reads using sirFAST with a maximum edit distance of 2 to the reference human genome assembly (GRCh37) with the paired-end mapping options. Then, we removed PCR duplicates and obtained a total of 2,558,246,235 (47.73%) reads that mapped to GRCh37. Using the CG mapping files, and applied the same parameters (filter out those mappings with $e > 2$; and PCR duplicate removal). We observed that 2,450,884,557 (45.7%) of the reads were mapped by the official CG mapper. However, we also note that the CG mapper can also map reads with more errors, and the raw mapping ratio for CG was 50.79%. We note that the mapping ratio compar-

[5] $Precision = \frac{TP}{TP+FP}$ where *TP* indicates the number of reads that map to the same position that was simulated and *FP* indicates the number of reads that map to a different position. However, the reads have the right insert size and orientation.

[6] $Recall = \frac{TP}{TP+FN}$ where *FN* indicates the number of reads that map to a different position than that was simulated and the insert size or the read orientation is not as expected.

Table 2
Mapping sensitivity, precision, and recall values for sirFAST using simulated data sets with different error thresholds. We mapped 500,000 read pairs, and compared their "best" mapping locations with their true locations as simulated from the reference genome. Note that all numbers given above are for number of read pairs.

Format	Data set	TP	FP	FN	Sensitivity (%)	Precision (PPV) (%)	Recall (%)
1	set 1 ($e = 0$)	481,379	17,528	1093	96.28	96.49	99.77
	set 2 ($e = 1$)	481,438	17,602	960	96.29	96.47	99.80
	set 3 ($e = 2$)	480,966	18,137	897	96.19	96.37	99.81
2	set 1 ($e = 0$)	489,158	10,842	0	97.83	97.83	100.00
	set 2 ($e = 1$)	488,142	11,775	62	97.62	97.64	99.98
	set 3 ($e = 2$)	487,793	12,099	74	97.55	97.57	99.98

Table 3
The comparison of sirFAST and CG mappers on the real data set (n = 5,360,139,478 reads). We considered the mapping positions with at most 2 errors for both sirFAST and the CG mapper.

Data set	sirFAST		CG mapper	
	Mapped	Mapping %	Mapped	Mapping %
NA12878	2,558,246,235	47.73	2,450,884,557	45.7

isons might be slightly different if other genomes are used for benchmarking. However, since the dataset we used is at extremely high depth (62.5X), we expect to observe only negligible differences in benchmarking other data sets. We also would like to point out that, the NA12878 genome was sequenced many times with many other HTS methods, which also lends itself for downstream analysis comparisons.

5. Conclusions

In this manuscript, we described the first publicly-available read mapper, *sirFAST*, designed to align reads generated by the Complete Genomics platform to the reference genome. The CG reads have very different properties when compared with data from other platforms. The reads are composed of several read sections where there may be overlaps or gaps in between. Our algorithms search for all possible combinations of expected gaps, while maintaining high sensitivity and mapping speed. Performance tests on both simulated and real data revealed high precision and recall rates in our mapping results with sirFAST.

To develop sirFAST, we made a number of design choices, such as using combined seeds and calculating Hamming distance over the seeds instead of implementing the Smith-Waterman [25] algorithm, and using hash tables for seed lookups instead of using suffix arrays. We note that different choices may affect the performance slightly, and in practice, some designs may increase or decrease performance. However, theoretically, full dynamic programming requires $O(n^2)$ run time, where our combined seed approach takes $O(kl)$ where k is the total gap size in the read, and l is the seed size ($l = 10$ in our dataset). [7] Similarly, we use hash tables for seed queries, which can be performed in $O(1)$ time, where suffix index requires binary search that takes $O(\log n)$ time.

Because of the relatively low cost of generating CG data, and because the CG data generation is provided as a service only, many researchers are now using the Complete Genomics service to analyze their genomes of interest. However, the lack of alternative analysis tools, from read mappers to SNP and SV callers, limit the discovery efforts to running a single set of proprietary algorithms developed by the Complete Genomics company. In fact, it is well known that different algorithms have different power in detection of genomic variants, especially structural variations [10, 28]. We hope and believe the open source implementation of our algorithms (sirFAST), which are introduced in this paper, will not only enable researchers and practitioners to more effectively use the Complete Genomics data sets but also pave the way to new research that can greatly improve the variation detection mechanisms for the Complete Genomics platform, thereby increasing the scientific community's ability to find important genomic variants.

[7] This run time estimation is calculated for the current read length and seed length, which may change if the read format drastically changes.

Acknowledgments

This study is supported by an NIH grant (HG006004) to C.A. and O.M. and a Marie Curie Career Integration Grant (PCIG-2011–303772) to C.A. under the Seventh Framework Programme. C.A. also acknowledges support from The Science Academy of Turkey, under the BAGEP program.

References

[1] J.O. Korbel, A.E. Urban, J.P. Affourtit, et al., Science 318 (5849) (2007) 420–426, http://dx.doi.org/10.1126/science.1149504. URL: http://dx.doi.org/10.1126/science.1149504.
[2] D.A. Wheeler, M. Srinivasan, M. Egholm, et al., Nature 452 (7189) (2008) 872–876, http://dx.doi.org/10.1038/nature06884. URL: http://dx.doi.org/10.1038/nature06884.
[3] M. Margulies, M. Egholm, W.E. Altman, et al., Nature 437 (7057) (2005) 376–380, http://dx.doi.org/10.1038/nature03959. URL: http://dx.doi.org/10.1038/nature03959.
[4] D.R. Bentley, S. Balasubramanian, H.P. Swerdlow, et al., Nature 456 (7218) (2008) 53–59, http://dx.doi.org/10.1038/nature07517. URL: http://dx.doi.org/10.1038/nature07517.
[5] K.J. McKernan, H.E. Peckham, G.L. Costa, et al., Genome Res. 19 (9) (2009) 1527–1541, http://dx.doi.org/10.1101/gr.091868.109. URL: http://dx.doi.org/10.1101/gr.091868.109.
[6] J. Eid, A. Fehr, J. Gray, et al., Science 323 (5910) (2009) 133–138, http://dx.doi.org/10.1126/science.1162986. URL: http://dx.doi.org/10.1126/science.1162986.
[7] M.L. Metzker, Nat. Rev. Genet. 11 (1) (2010) 31–46, http://dx.doi.org/10.1038/nrg2626. URL: http://dx.doi.org/10.1038/nrg2626.
[8] N.A. Fonseca, J. Rung, A. Brazma, J.C. Marioni, Bioinformatics 28 (24) (2012) 3169–3177, http://dx.doi.org/10.1093/bioinformatics/bts605. http://dx.doi.org/10.1093/bioinformatics/bts605.
[9] R. Nielsen, J.S. Paul, A. Albrechtsen, Y.S. Song, Nat. Rev. Genet. 12 (6) (2011) 443–451, http://dx.doi.org/10.1038/nrg2986. URL: http://dx.doi.org/10.1038/nrg2986.
[10] C. Alkan, B.P. Coe, E.E. Eichler, Nat. Rev. Genet. 12 (5) (2011) 363–376, http://dx.doi.org/10.1038/nrg2958. URL: http://dx.doi.org/10.1038/nrg2958.
[11] K.R. Bradnam, J.N. Fass, A. Alexandrov, et al., Gigascience 2 (1) (2013) 10, http://dx.doi.org/10.1186/2047-217X-2-10. URL http://dx.doi.org/10.1186/2047-217X-2-10.
[12] R. Drmanac, A.B. Sparks, M.J. Callow, et al., Science 327 (5961) (2010) 78–81, http://dx.doi.org/10.1126/science.1181498. URL: http://dx.doi.org/10.1126/science.1181498.
[13] P. Carnevali, J. Baccash, A.L. Halpern, et al., J. Comput. Biol. 19 (3) (2012) 279–292, http://dx.doi.org/10.1089/cmb.2011.0201. URL: http://dx.doi.org/10.1089/cmb.2011.0201.
[14] Genomes Project Consortium, Nature 491 (7422) (2012) 56–65, http://dx.doi.org/10.1038/nature11632. URL: http://dx.doi.org/10.1038/nature11632.
[15] M. Burrows, D.J. Wheeler, A Block Sorting Lossless Data Compression Algorithm, Technical Report, Digital Equipment Corporation, 1994.
[16] C. Alkan, J.M. Kidd, T. Marques-Bonet, et al., Nat. Genet. 41 (10) (2009) 1061–1067, http://dx.doi.org/10.1038/ng.437. URL: http://dx.doi.org/10.1038/ng.437.

[17] F. Hach, F. Hormozdiari, C. Alkan, et al., Nat. Methods 7 (8) (2010) 576–577, http://dx.doi.org/10.1038/nmeth0810-576. URL: http://dx.doi.org/10.1038/nmeth0810-576.
[18] F. Hormozdiari, F. Hach, S.C. Sahinalp, E.E. Eichler, C. Alkan, Bioinformatics 27 (14) (2011) 1915–1921, http://dx.doi.org/10.1093/bioinformatics/btr303. URL: http://dx.doi.org/10.1093/bioinformatics/btr303.
[19] H. Xin, D. Lee, F. Hormozdiari, et al., Accelerating Read Mapping with Fasthash., BMC Genomics 14 (Suppl. 1), 2013, S13. doi:10.1186/1471-2164-14-S1-S13. URL: <http://www.biomedcentral.com/1471-2164/14/S1/S13>
[20] H. Li, B. Handsaker, A. Wysoker, et al., Bioinformatics 25 (16) (2009) 2078–2079. URL: <http://samtools.sourceforge.net>.
[21] F. Hormozdiari, C. Alkan, E.E. Eichler, S.C. Sahinal, Genome Res. 19 (7) (2009) 1270–1278, http://dx.doi.org/10.1101/gr.088633.108. URL: http://dx.doi.org/10.1101/gr.088633.108.
[22] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, D.J. Lipman, J. Mol. Biol. 215 (3) (1990) 403–410. doi:10.1016/S0022-2836(05)80360-2, URL: http://dx.doi.org/10.1016/S0022-2836(05)80360-2.
[23] B. Ma, J. Tromp, M. Li, Bioinformatics 18 (3) (2002) 440–445.
[24] V. Levenshtein, Soviet Phys. Doklady 10 (8) (1966) 707–710.
[25] T.F. Smith, M.S. Waterman, J. Mol. Biol. 147 (1) (1981) 195–197.
[26] E. Ukkonen, On approximate string matching, in: Proceedings of the international FCT-conference on fundamentals of computation theory, 1983, pp. 487–495.
[27] R.W. Hamming, Bell Syst. Tech. J. 26 (2) (1950) 147–160.
[28] R.E. Mills, K. Walter, C. Stewart, et al., Nature 470 (7332) (2011) 59–65. doi:10.1038/nature09708. URL: http://dx.doi.org/10.1038/nature09708.